Real-Time Procedural Terrain Generation with Marching Cubes

Joseph Chambers-Graham

Abstract

This project explores a method for procedurally generating terrain by applying the a variation of the Marching Cubes algorithm known as the Transvoxel algorithm. We use an octree data structure to break down a large world into chunks at varying levels of detail, and apply the parallel processing power of the GPU to rapidly generate geometry on a per-chunk basis. We then explore applications of this approach, modifying the geometry in real-time by making localized changes to the underlying distance function. Finally, we use the meshes we have generated with a well-known physics library.

Contents

1 Introduction

- project and report overview

2 Background

2.1 Signed Distance Functions

At the core of the Marching Cubes algorithm, and the way we will be defining shapes, is the signed distance function (SDF). This is a function of the form $f : \mathbb{R}^3 \to \mathbb{R}$. The shape represented by an SDF is the implicit surface $f(x, y, z) = 0$. Furthermore, it is desirable for an SDF to have the following properties:

i. At all points where $f(x, y, z) \neq 0$, if (x, y, z) is inside the surface, $f(x, y, z) < 0$. If (x, y, z) is outside the surface, $f(x, y, z) > 0$. This is the most important property of an SDF, without which the Marching Cubes algorithm is unlikely to produce valid geometry.

ii. At all points where $f(x, y, z) \neq 0$, the value $f(x, y, z)$ should be the smallest (signed) euclidean distance from the point (x, y, z) to the surface $f(x, y, z) = 0$. In practice, many implicit functions we are using do not have this property, however for best results the value $f(x, y, z)$ should be a good approximation of the actual distance, and for floating point precision reasons, must be at least the same order of magnitude. This property will be used by the Marching Cubes algorithm to interpolate the positions of vertices, and as such a better distance approximation will lead to a more accurate mesh representation of the SDF. An SDF such that $f(x, y, z)$ gives the correct distance everywhere is an *exact* SDF. Otherwise, it is an *approximate* SDF.

iii. f is continuous everywhere, and has all partial derivatives. This is useful for calculating the normal to the surface at a given point. In practice, we will relax these restrictions, but this should still hold in places where the algorithm might generate vertices.

Many geometric shapes have exact SDF representations. Figure 1 shows a 2 dimensional exact SDF for a circle.

However, many useful shapes, in particular heightmaps and shapes those defined by random noise, are not easy to represent as an exact SDF. For this sort of shape, we use an approximate SDF. Figure 2 shows a 2 dimensional approximate SDF for the heightmap defined by $y = 1 - \frac{x^2}{3}$. It is of course possible to define the exact distance function for any implicit shape, however such problems are often minimisation problems with with solutions that are expensive to compute. For example,even this simple example computing the exact distance to a quadratic curve requires computing the roots of a cubic polynomial, and computing the exact distance to a cubic curve requires solving a polynomial of degree 5, for which there is no elementary solution. Since this

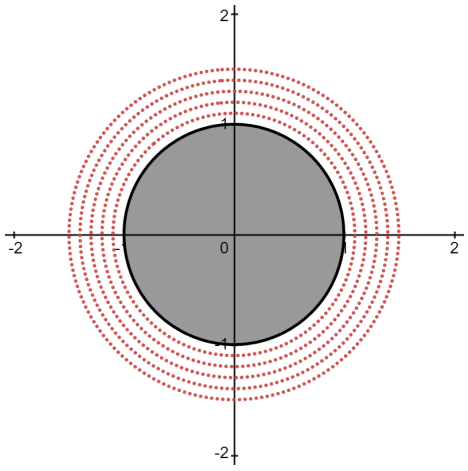

Figure 1: Example of a 2 dimensional SDF representing a circle. The function shown is $f(x,y) = \sqrt{x^2+y^2} - 1$, with the area where $f(x,y) < 0$ shaded. Also shown are the contours where $f(x,y) = 0.1, 0.2, ...0.5$

is useful for defining shapes based on interpolation splines, it is certainly worth considering when an approximation may be worth using due to the reduced computational cost.

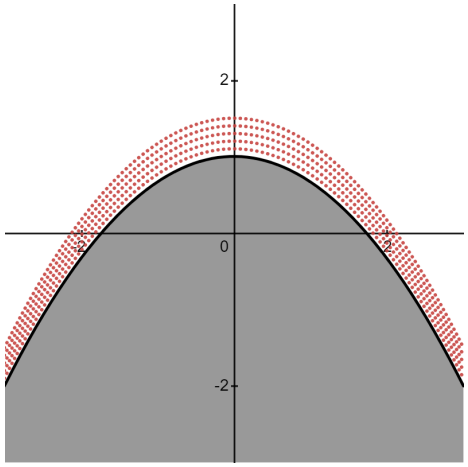

Figure 2: Plot of the SDF $f(x,y) = y - \left(1 - \frac{x^2}{3}\right)$. Note the contour lines are no longer uniformly spaced, as they would be with an exact SDF. In fact, the distance approximation gets worse, as the slope of the surface increases.

The basic set-theoretical operations of union, intersection, and difference have equivalent representations using the min and max functions. At this point, one many note that the function $\min(f,g)$ may fail to be differentiable at the point where $f = g$. Here, we may simply take the derivative of f or g.
It is worth noting that some literature chooses the terminology of signed *density* functions. This is equivalent to an approximate SDF, however the sign convention is flipped, so that a negative value is outside the surface.

2.2 Noise

A typical noise function takes as input a point in $\mathbb{R}^n$ and returns a pseudorandom value, usually between 0 and 1. The use of noise to generate convincing heightmaps is well-documented, and typical implementations are as a function of the form $y = h(x, z)$ giving the height of the terrain at a given (x, z) coordinate. This can be simply extended to an approximate SDF, using the formula $f(x, y, z) = y - h(x, z)$. Note that this is in fact an approximate SDF, and that the distance approximation worsens as the steepness of the slope of $h(x, z)$ increases. This phenomenon can be seen in figure 2.
The use of 3 dimensional noise as a distance function is also possible, and yields interesting results.

2.3 Marching Cubes Algorithm

- explain algorithm - basically just a reference?

3 Design

3.1 GPU Implementation

- How is the algorithm modified
- speed comparison (GPU is much faster)

3.2 LOD system

- cannot render everything at max resolution

3.2.1 Octree

- Chunks with same point count, but scaled based on depth of octree

3.3 Transvoxel Algorithm

3.3.1 The problem it solves

- cracks in between chunks at different LOD

3.3.2 How the algorithm works

- basically just a reference again
- reference implementation is complicated so could do with explanation

3.3.3 Adaption of the algorithm to GPU

- optimizations to make use of sequential generation have actually been removed
- regardless, this is still pretty fast on the GPU
- similar approach to parallel marching cubes

3.3.4 More issues

- zero width triangles
- edge cases where it still goes wrong (maybe a way to fix this by revisiting the octree)

3.4 Terrain Modification

3.4.1 Strategy

- adding or removing primitives using SDF operations
- regenerating chunks
- when to regenerate chunks, and which ones
- how many primitives is reasonable before slowdown due to SDF evaluation?
- bounding boxes of primitives to increase this limit - dont need to evaluate parts not affected

3.5 Physics

- Just use bullet physics - well-known and optimized library
- zero-width triangles havent turned out to be a problem so far
- generating physics meshes is slow - need to find a solution to this
- In fact, physics is currently the *slowest* part of this...

3.5.1 SDF-based physics

- perhaps look at this - might be a struggle because of non-exact SDFs, and how would we do it for anything that isn't a sphere anyway?

4 Implementation

- C++ with fairly low-level opengl libraries
- key code probably in design section already
- section for the "boring bits" of the implementation, rather than key ideas
- perhaps section to explain other things that needed bugfixes?
- something tying it all together?

5 Conclusion

- demonstration of final product
- conclusion
- somehting about future work?